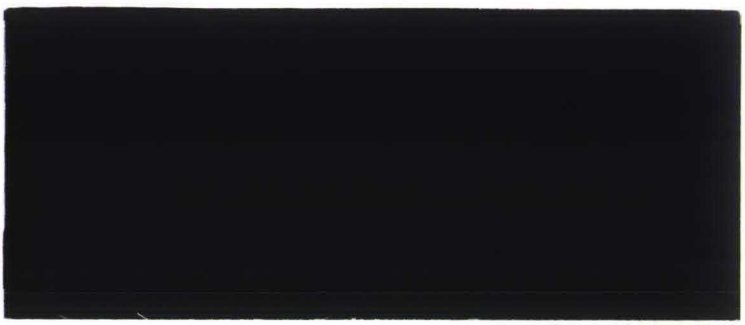
Vector computers, Monte Carlo simulation, and regression analysis

Kleijnen, J.P.C.; Annink, B.

Publication date:
1990

UNIVERSITEIT BRABANT
POSTBOX 90153
5000 LE TILBURG
THE NETHERLANDS



DEPARTMENT OF ECONOMICS

RESEARCH MEMORANDUM

VECTOR COMPUTERS, MONTE CARLO SIMULATION, AND REGRESSION ANALYSIS: AN INTRODUCTION

Jack P.C. Kleijnen and Ben Annink

FEW 443

Kenmerk: 320.90.131

VECTOR COMPUTERS, MONTE CARLO SIMULATION, AND REGRESSION ANALYSIS: AN INTRODUCTION

Jack P.C. Kleijnen
and
Ben Annink

Version 2: May 1990
Version 1: August 1989

Correspondence should be directed to: Prof. J.P.C. Kleijnen, Department of Information Systems and Auditing, School of Business and Economics, Katholieke Universiteit Brabant (Tilburg University), 5000 LE Tilburg, Netherlands. FAX: 013-663019. E-mail: kleijnen@kub.nl.

VECTOR COMPUTERS, MONTE CARLO SIMULATION, AND REGRESSION ANALYSIS: AN INTRODUCTION

Jack P.C. Kleijnen

and

Ben Annink

Department of Information Systems and Auditing, School of Business and Economics, Katholieke Universiteit Brabant (Tilburg University), 5000 LE Tilburg, Netherlands. Fax: 013-663072. E-mail: kleijnen@kub.nl

Vector computers provide a new tool for management scientists. The application of that tool requires thinking in vector mode. This mode is examined in the context of Monte Carlo experiments with regression models; these regression models serve as metamodels in simulation experiments. The vector mode needs to exploit a specific dimension of the Monte Carlo experiment, namely the replicates of the Monte Carlo experiment. Taking advantage of the machine architecture gives code that computes Ordinary Least Squares estimates on a Cyber 205 in 2% of the time needed on a Vax 8700. For Generalized Least Squares estimates, the code runs slower on the Cyber 205 than on the VAX, if the regression model is small; for large models the CYBER 205 runs much faster.

(SUPERCOMPUTERS; DISTRIBUTION SAMPLING; MULTIVARIATE DISTRIBUTION; COMMON SEEDS; METAMODEL)

1. Introduction

This paper illustrates three important points about the new generation of computers called "supercomputers":

(i) Efficient supercomputing requires adjusting algorithms to take advantage of the specific architecture of the computing hardware.

(ii) Expensive supercomputers may be slower than general purpose machines on problems for which they are not suited.

(iii) The increased speed of the supercomputing calculation may not outweigh the burden of constructing the specialized code: the researcher's time is valuable too.

This paper focuses on the use of supercomputers in Monte Carlo experiments with regression analysis. So this paper may be of interest to management scientists for several reasons:

(i) Regression analysis is often used by management scientists to analyze simulation data and real-world data. The role of regression analysis in simulation will be explained in § 2.

(ii) The study shows how supercomputers can be applied in Monte Carlo experiments. Monte Carlo experiments are related to stochastic discrete event dynamic simulation: both methods use pseudorandom numbers, but Monte Carlo experiments are static whereas simulation models are dynamic (a case in point is a queuing simulation); see Teichroew (1965). So Monte Carlo experiments are simpler. Our study may challenge other researchers to apply supercomputers to Monte Carlo and simulation models.

There are several types of supercomputers: vector computers should be distinguished from traditional scalar computers and truly parallel computers. Traditional computers such as the IBM 370 and the VAX series, execute one instruction after the other; so they operate sequentially. Truly parallel computers such as the HYPERCUBE, have many Central Processing Units (CPU's) that can operate independently of each other; this is called coarse grain parallelism. Vector computers such as the CRAY 1 and the CYBER 205, have a "vector processing" capability: fine grain parallelism. Consider, for example, the computation of the inner product of two vectors: $\mathbf{v}_1' \mathbf{v}_2 = \Sigma_{j=1}^{n} v_{1j} v_{2j}$. This computation requires n identical scalar operations $v_{1j}\ v_{2j}$. The vector processor starts computing $v_{1j}\ v_{2j}$ while the computation of the predecessors $v_{1(j-1)}\ v_{2(j-1)}$, $v_{1(j-2)}\ v_{2(j-2)}$, $\cdots$

is still in process! So a vector computer works as an assembly line. A technical condition is that the scalar operations do not depend on each other; in the example the computation of the scalar product $v_{1j}\ v_{2j}$ does not need the other scalar products, especially the predecessors $v_{1(j-1)}$ $v_{2(j-1)}$ through $v_{11}\ v_{21}$. The architecture of a vector computer is called a pipeline. The pipeline or assembly line requires a fixed set up cost; consequently a vector computer works efficiently only if a "large" number of identical (scalar) operations can be executed independently of each other. In the example, n must be large (a rule of thumb is $n \geq 50$; we shall return to this issue). This paper's main issue is: how can we formulate the Monte Carlo model such that a vector computer can be applied efficiently? We do not discuss the use of truly parallel computers in simulation but refer to Heidelberger (1988). Technical details on the new generation of "supercomputers" are given by Levine (1982).

Our paper is organized as follows. In § 2 we summarize the well-known linear regression model and its application in simulation experiments with common pseudorandom numbers. This regression model is studied in a Monte Carlo experiment. In § 3 we show how the Monte Carlo program can be vectorized: we discover a "third dimension" of Monte Carlo experiments. § 4 gives numerical results. § 5 gives conclusions. References and appendices complete the paper.

2. Regression Models and Simulation

Consider the well-known linear regression model

$$E(\mathbf{y}) = \mathbf{X}\,\underline{\beta} \tag{2.1}$$

with $\mathbf{y} = (y_1,\ldots,y_i,\ldots,y_n)'$, $\underline{\beta} = (\beta_1,\ldots,\beta_j,\ldots,\beta_Q)'$ and $\mathbf{X} = (x_{ij})$ where $i = 1,\ldots,n$ and $j = 1,\ldots Q$. We assume additive errors $\mathbf{e} = (e_1,\ldots,e_i,\ldots,e_n)'$ (the errors are also called disturbance or noise):

$$\mathbf{y} = \mathbf{X}\,\underline{\beta} + \mathbf{e}. \tag{2.2}$$

We further assume that $\mathbf{e}$ is n-variate normally (N_n) distributed:

$$\mathbf{e} \sim N_n [0_n, \mathbf{cov}(\mathbf{e})], \tag{2.3}$$

where 0_n denotes a column of n zeros; $\mathbf{cov}(\mathbf{e})$ denotes the variance-covariance matrix of $\mathbf{e}$; $\mathbf{cov}(\mathbf{e})$ equals $\mathbf{cov}(\mathbf{y})$ because of (2.2); $\mathbf{cov}(\mathbf{y})$ is assumed to be nonsingular.

This regression model can be applied to analyze the results of a simulation experiment. We first consider a simplistic simulation experiment, namely an M/M/1 queue with a fixed arrival rate of (say) one and n different service rates μ_i ($i = 1,\ldots,n$). The n simulation responses are the average waiting times $\bar{w}_i$ ($= \Sigma_{t=1}^{T} w_t(\mu_i)/T$ assuming T customers are simulated for each μ_i). Within the experimental area defined by $\min_i \mu_i \leq \mu \leq \max_i \mu_i$ we may model the response curve $\bar{w} = f(\mu)$ by the second order approximation $\bar{w} = \alpha_0 + \alpha_1 \mu + \alpha_2 \mu^2$; in the notation of (2.1) we have $y = \bar{w}$, $\beta_1 = \alpha_0$, $x_1 = 1$, $\beta_2 = \alpha_1$, $x_2 = \mu$, $\beta_3 = \alpha_2$, $x_3 = \mu^2$. Such an approximation is called a regression metamodel, since it is a model of the input/output behavior of the underlying simulation model; see Kleijnen (1987) for details including realistic examples.

In the M/M/1 example we may simulate the n queuing systems defined by the n service rates μ_i, while using the *same* pseuorandom number sequence. In the simplistic example it suffices to use the same seed for the pseudorandom number generator. In realistic examples, the variance reduction technique of common random numbers may require extra care if we wish to create strong positive correlations between the simulation responses. Readers familiar with Schruben and Margolin's strategy for assigning random number streams, will know that some elements of $\mathbf{cov}(\mathbf{e})$ are negative (when antithetic pseudorandom numbers are used). Anyhow, $\mathbf{cov}(\mathbf{e})$ in (2.3) may be nondiagonal; also see Kleijnen (1988).

Next we extend the M/M/1 example a bit: suppose we wish to study not only the effect of the service rate μ, but also the effect of the priority rule. Until now that rule was implicitly first-in-first-out (FIFO). Suppose we also examine an alternative rule, say short-jobs-first (SJF). Suppose further that we extend the queuing model such that S servers are simulated with (say) S = 1,2,3,4. Then we have three "factors" in the simulation experiment: service rate, priority rule, and number of servers. The statistical theory of *experimental design* can help us to decide which combinations of factor "levels" or "values" to simulate. That

theory assumes a regression metamodel! Suppose we assume that the response surface can be modeled by a first order approximation: $y = \beta_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4$ with $y = \bar{w}$, $x_1 = 1$; $x_2 = -1$ if $\mu = \min_i \mu_i$ and $x_2 = +1$ if $\mu = \max_i \mu_i$; $x_3 = -1$ if S=1 and $x_3 = +1$ if S=4; $x_4 = -1$ if FIFO applies and $x_4 = +1$ if SJF holds. Then we can estimate the four regression parameters if we simulate only $8 = 2^{3-1}$ combinations of these three factors; see Table 1. We point out that the four column vectors $\mathbf{x}_j$ (j = 1,2,3,4) are mutually orthogonal; $\mathbf{x}_1$ is a constant, not a factor. For a second order approximation, design theory gives tables analogous to Table 1, albeit that more than two levels per factor must be included and that some columns are not orthogonal. There is a vast literature on experimental design; Kleijnen (1987) gives designs that are particularly useful in simulation experiments.

TABLE 1

Experimental Design for Three Factors

Combination i	Factor levels x_2	x_3	x_4
1	-1	-1	+1
2	+1	-1	-1
3	-1	+1	-1
4	+1	+1	+1
5	-1	-1	+1
6	+1	-1	-1
7	-1	+1	-1
8	+1	+1	+1

We assume that in the regression analysis of simulation data, the matrix of independent regression variables **X** in (2.1) follows from an experimental design for k factors: $\mathbf{D} = (d_{ih})$ with h = 1,...,k. Table 1 specifies such a matrix **D** for k = 3 and n = 8. The number of regression

parameters is denoted by Q; in the example of Table 1 we have Q = 4. In a second order approximation we have $Q = 1 + k + k(k-1)/2 + k$. In a well designed simulation experiment it is easy to *replicate* each factor combination; that is, row i of **X** or **D** can be observed $m_i \geq 2$ times. So a terminating simulation is repeated with m_i independent pseudorandom number streams; in non-terminating or steady-state simulations m_i subruns may be obtained; see Kleijnen (1987, pp. 8-10, 63-83).

Earlier we mentioned the use of common random numbers. This means that the n simulated systems (combinations of factor levels) use the same seed. This common seed yields better comparisons among the n simulation responses. However, that seed may happen to create an outlier. Therefore we repeat the simulation experiment a number of times, namely m times (so m_i of the preceding paragraph reduces to a constant m).

The remainder of this paper focuses on the regression model specified by (2.1) through (2.3). As we said in the introduction (§ 1), regression analysis is often used by management scientists to analyze simulation data (as we just explained) and real-world data. Originally we wished to examine different estimators of the regression parameters $\underline{\beta}$ and different tests for validating the fit of the resulting regression model to the data (simulated or real). For that study we use Monte Carlo simulation: we select X, $\underline{\beta}$, cov(**e**) and m; next we generate responses **y**; these data yield $\underline{\beta}$ estimators; these estimators are compared with the true parameter vector $\underline{\beta}$ (which is known in the Monte Carlo experiment); this comparison is repeated (say) L = 100 times to obtain reliable Monte Carlo results. That whole experiment is reported in Kleijnen (1990). We wished to use a vector computer for that experiment. However, it turned out that a vector computer may be inefficient in this case. The present paper explains why this is so. So we concentrate on those aspects of the original experiment that we need in order to explain the use of vector computers in Monte Carlo experiments with regression models applied to simulation data, obtained by a sound experimental design.

Table 2 summarizes the data that are available to estimate the regression parameters $\underline{\beta}$ (this table is reproduced from Kleijnen, 1988, p. 66). The responses y_{ir} yield the following unbiased estimators of

$\sigma_{ih} = \mathrm{cov}(y_i, y_h) = \mathrm{cov}(y_{ir}, y_{hr})$ where y_{ir} is the r^{th} replication of the i^{th} factor combination:

$$\hat{\sigma}_{ih} = \frac{\sum_{r=1}^{m} (y_{ir}-\bar{y}_i)(y_{hr}-\bar{y}_h)}{m-1} = \left[\sum_{r=1}^{m} y_{ir}y_{hr} - \bar{y}_i\bar{y}_h m)\right]/(m-1)$$

$$(i,h = 1,\ldots,n) \quad (m \geq 2), \tag{2.4}$$

with the averages $\bar{y}_i = \sum_{r=1}^{m} y_{ir}/m$; by definition we have $\sigma_{ii} = \sigma_i^2$. We refer to Neely (1966) for a discussion of the different numerical accuracies of the two expressions in (2.4). In matrix notation the last expression in (2.4) becomes

$$\hat{\mathbf{cov}}(\mathbf{y}) = (\mathbf{Y}\,\mathbf{Y}' - \bar{\mathbf{y}}\,\bar{\mathbf{y}}'m)/(m-1), \tag{2.5}$$

with $\hat{\mathbf{cov}}(\mathbf{y}) = (\hat{\sigma}_{ih})$, $\mathbf{Y} = (y_{ir})$ and $\bar{\mathbf{y}} = (\bar{y}_i)$. It is simple to prove that $\hat{\mathbf{cov}}(\mathbf{y})$ is singular for $m \leq n$.

TABLE 2

Regression Data

Combination i (effects: $\beta_1 \ldots \beta_j \ldots \beta_Q$)	Responses y_{ir} (seed 1)...(seed r)...(seed m)	Average response $\bar{y}_i$	Estimated (co)variances $\hat{\sigma}_{ih}$
$x_{11} \ldots x_{1j} \ldots x_{1Q}$	$y_{11} \quad \ldots \quad y_{1r} \quad \ldots \quad y_{1m}$	$\bar{y}_1$	$\hat{\sigma}_1^2 \hat{\sigma}_{12} \ldots \hat{\sigma}_{1n}$
$x_{i1} \ldots x_{ij} \ldots x_{iQ}$	$y_{i1} \quad \ldots \quad y_{ir} \quad \ldots \quad y_{im}$	$\bar{y}_i$	$\hat{\sigma}_i^2 \ldots \hat{\sigma}_{in}$
$x_{n1} \ldots x_{nj} \ldots x_{nQ}$	$y_{n1} \quad \ldots \quad y_{nr} \quad \ldots \quad y_{nm}$	$\bar{y}_n$	σ_n^2

We consider two different point estimators for the regression parameters $\underline{\beta}$. The simple classic estimator uses *Ordinary Least Squares* or OLS:

$$\hat{\underline{\beta}} = (X'X)^{-1}X'\underline{\bar{y}} \ , \tag{2.6}$$

which assumes $n \geq Q$ and rank $(X) = Q$. If $\mathbf{cov}(y)$ were known, then a better estimator would use *Generalized Least Squares* (GLS). Since $\mathbf{cov}(y)$ is unknown in practice, we may replace it by the estimator $\hat{\mathbf{cov}}(y)$ of (2.5), and use *Estimated Generalized Least Squares* or EGLS:

$$\tilde{\underline{\beta}} = (X'\hat{\mathbf{cov}}(y)^{-1}X)^{-1}X\ \hat{\mathbf{cov}}(y)^{-1}\underline{\bar{y}} \ , \tag{2.7}$$

which assumes that $\hat{\mathbf{cov}}(y)$ is non-singular; also see (2.3) and Dijkstra (1970).

3. Vectorizing the Monte Carlo program

We wish to construct a Monte Carlo program to compare the OLS and EGLS estimates of (2.6) and (2.7). We might use the vector mode to compute an individual element y_{ir} of $\mathbf{Y} = (y_{ir}) = \mathbf{X}\ \underline{\beta} + \mathbf{e}$; see Table 2 and (2.2). X is n x Q; typically n and Q range between n = 8 and Q = 4 (see Table 1) and n = 32 and Q = 22 (see Table 3 later on). It is well-known that vector computers are inefficient if the number of parallel operations is "small", say, smaller than 50; see Levine (1982) and SARA (1984). So it is inefficient to vectorize the computation of an individual y_{ir}.

Next we consider the vector computation of either the rows or the columns of Table 2. Since there are only n rows (factor combinations), vectorization is again inefficient. Because the columns of Table 2 are statistically independent (see § 2), vectorization is possible. Since simulation replication is expensive, m will be small in practice; the minimum is $m = n + 1$ (otherwise, $\hat{\mathbf{cov}}(y)$ is singular). So vectorizing the columns of Table 2 is also inefficient.

3.1 The third dimension

The Monte Carlo experiment is replicated L = 100 times (see § 2). We speak of Monte Carlo replicates ℓ with $\ell = 1,\ldots,L$, which must be distinguished from the simulation replicates $r = 1,\ldots,m$. The L Monte Carlo replicates are statistically independent; they can be vectorized as we shall see. The more of these replicates we wish to obtain, the more efficient the vector computer becomes. We may visualize our problem as follows. There is a three-dimensional box to be filled in parallel with errors $e_{ir\ell}$ with $i = 1,\ldots,n$; $r = 1,\ldots,m$; $\ell = 1,\ldots,L$. This box is filled in steps 1 through 3 below. In step 4 statistics such as $\hat{\mathbf{cov}}(\mathbf{y})$ are computed.

Step 1: Sample pseudorandom numbers

Kleijnen (1989) evaluates several procedures for the parallel generation of pseudorandom numbers $x \sim U(0,1)$. Kleijnen and Annink (1989) recommend the following generator. Take a scalar multiplicative congruential generator with a multiplier that gives acceptable statistical behavior; see Park and Miller (1988). To initialize the vector version of this generator, first generate - in scalar mode - a vector of J successive pseudorandom integers $\mathbf{x} = (x_0, x_1, x_2, \ldots, x_{J-2}, x_{J-1})'$ with seed x_0 and $x_j = (a\ x_{j-1})$ mod m for $j = 1,2,\ldots, J-1$. To obtain numbers between zero and one, divide by m. Once and for all compute a scalar multiplier (a^J) mod m. Vector multiplication of the vector $\mathbf{x}$ with this scalar multiplier gives a new vector: $(x_J, x_{J+1}, \ldots, x_{2J-2,2J-1})'$. In this way the pseudorandom numbers are generated in parallel and yet in exactly the same order as they would have been produced in scalar mode. At the end of the Monte Carlo experiment the vector of the last J numbers should be stored, so that the experiment may be continued later on.

We mentioned that vector computers become more efficient as the number of parallel operations increases. For the CYBER 205, however, there is a technical upper limit: $J = 2^{16} - 1 = 65{,}535$ (since this computer uses 16 bits for addressing; see SARA, 1984, p. 26).

There is a computational problem: overflow occurs when computing (a^J) mod m. This problem is solved, using the computer science techniques

of controlled integer overflow and the CYBER 205's two's complement representation of negative integers; the computer program in appendix 1 gives technical details; also see Park and Miller (1988).

So we generate a vector of J pseudorandom numbers. We store that vector, which is then available to fill the n x m x L box.

Step 2: Sample independent standard normal variates

There are several techniques for generating normal variates; see Devroye (1986). We take a procedure that fits a vector computer:

$$z_1 = (-2 \ln x_1)^{\frac{1}{2}} \cos 2\pi x_2 \qquad (3.1.a)$$

$$z_2 = (-2 \ln x_1)^{\frac{1}{2}} \sin 2\pi x_2, \qquad (3.1.b)$$

where the mutually independent pair x_1 and x_2 with $x \sim U(0,1)$ yields the mutually independent pair z_1 and z_2 with $z \sim N(0,1)$. To compute the functions $\ln$, cos and sin for a *vector* of numbers, we use FORTRAN 200's vector functions VLN, VCOS, and VSIN. Given a vector of L independent pseudorandom numbers x, we use the first half to compute L/2 independent parallel realizations of $\ln x_1$, and the second half to compute cos $(2\pi x_2)$ and sin $(2\pi x_2)$: Figure 1 gives a pseudo-FORTRAN program where π is computed through the arccosine function; see SARA (1984, p. 13). To convert this pseudo-FORTRAN into a FORTRAN 200 program, we can replace DO loops by the special syntax of FORTRAN 200; the supercomputer can also automatically translate the FORTRAN program of Figure 1 provided we add CONTINUE statements; see CDC (1986), SARA (1984, p. 17).

Note that Petersen (1988) generates z in parallel, not using (3.1.a) and (3.1.b), but Teichroew's procedure described in Naylor et al. (1966, p. 94).

Above we saw that we wish to fill a three-dimensional "box" with $e_{ir\ell}$. So we store the *vectors* z (with L elements) of Figure 1 into a three-dimensional *array* $Z(i,r,\ell)$.

FIGURE 1

Parallel computation of L variates z ~ N(0,1).

```
      L2 = L/2; PI = ACOS(-1.0); C = 2 * PI
      DO   20   LL = 1, L2
20         HELP1(LL) = SQRT(-2 * LOG(X(LL)))
      DO   30   LL = 1, L2
           HELP2(LL) = COS(X(LL + L2) * C)
           HELP3(LL) = SIN(X(LL + L2) * C)
           Z(LL) = HELP1(LL) * HELP2(LL)
30         Z(L2 + LL) = HELP1(LL) * HELP3(LL)
```

Step 3: Sample n-variate normally distributed variates

The errors within a column of Table 2 are statistically dependent: they are n-variate normal. We first consider a computer program for n = 2. In that case we sample the independent univariate standard normal variates z_1 and z_2, and compute the linear transformations $e_1 = \sigma_1 z_1$ and $e_2 = \sigma_2(\rho\, z_1 + (1-\rho^2)^{\frac{1}{2}} z_2)$ where $\rho = \sigma_{12}/(\sigma_1\sigma_2)$. Next we consider the general case. The sampling subroutine for multivariate normal **e** with covariance matrix **cov(e)** is

$$\mathbf{e} = \mathbf{C}\,\mathbf{z}, \tag{3.2}$$

with $\mathbf{z} = (z_1,\ldots,z_i,\ldots,z_n)'$ and independent $z_i \sim N(0,1)$, and **C** a lower triangular matrix defined by

$$\mathbf{C}\,\mathbf{C}' = \mathbf{cov}(\mathbf{e}). \tag{3.3}$$

C is computed by Choleski's technique; see Naylor et al. (1966. pp. 97-99) and standard software libraries such as IMSL and NAG. Once **C** is computed, we generate **e** through the linear transformation (3.2) of **z**. That transformation is not vectorized because n is too small.

To obtain M observations and L Monte Carlo replicates of **e**, we might apply the naive FORTRAN program of Figure 2, where M denotes the

maximum value of m in the experiment (here M = 33; see Table 3) and E(I,R,LL) is zero initially. Note that **C** or C(I,J) does not vary over seeds (R) and Monte Carlo replicates (LL); it does vary over the Monte Carlo experiments defined by **cov(y)**.

To vectorize this naive program we should make the *inner* DO loop long; therefore we move the LL loop; moreover we should store the columns of the array columnwise; see SARA (1984, pp. 15, 20-21, 33). These two guidelines yield Figure 3. (Note that the inner loop forms a so-called "linked triad"; hence it can be vectorized; see SARA, 1984, pp. 18-19.)

FIGURE 2

Naive FORTRAN program for **e**.

```
      DO    10    LL = 1,L
         DO    10    R = 1,M
            DO    10    I = 1,N
               DO    10    J = 1,I
10                E(I,R,LL) = E(I,R,LL) + C(I,J) * Z(J,R,LL)
```

FIGURE 3

Vectorized FORTRAN program for **e**.

```
      DO    20    I = 1,N
         DO    20    J = 1,I
            DO    20    R = 1,M
               DO    20    LL = 1,L
20                E(LL,R,I) = E(LL,R,I) + C(I,J) * Z(LL,R,J)
```

We point out that m and n vary with the Monte Carlo experiments. So an experiment may use only part of the pseudorandom numbers stored in the "box" E(LL,R,I). Implementing Figure 3 not only saves computer time, but it also runs experiments with common seeds (since all experiments pull pseudorandom numbers from the same box).

Note that we could generate M*L (instead of L) elements in parallel, if we replaced two loops - namely the loops for R and LL - in Figure 3 by a single loop - namely LR = 1,..., M*L - which would yield the two-dimensional array E(LR,I) of Figure 4. Then, however, we would have to rearrange this array into the three-dimensional array E(LL,R,I), because the latter array is needed for the computation of statistics such as $\hat{\text{cov}}(\mathbf{y})$, as we shall see next.

FIGURE 4

Alternative Vectorized Fortran program for **e**.

```
      ML = M*L
      DO  20   I = 1,N
        DO  20    J = 1,I
          DO  20     LR = 1,ML
20          E(LR,I) = E(LR,I) + C(I,J) * Z(LR,J).
```

Step 4: Compute statistics $\hat{\text{cov}}(\mathbf{y})$, $\hat{\underline{\beta}}$ *and* $\tilde{\underline{\beta}}$

Once we have the three-dimensional array **E**, we can easily compute estimates such as $\hat{\text{cov}}(\mathbf{y})$ defined in (2.5). This equation can also be computed as

$$\hat{\text{cov}}(\mathbf{y}) = \mathbf{e}\,\mathbf{e}'/(m-1) - \bar{\mathbf{e}}\,\bar{\mathbf{e}}'m/(m-1), \qquad (3.4)$$

with $\bar{\mathbf{e}} = (\bar{e}_1,\ldots,\bar{e}_i,\ldots,\bar{e}_n)'$ and $\bar{e}_i = \Sigma_{r=1}^{m} e_{ir}/m$. Figure 5 shows the vectorizable FORTRAN program for the computation of $\bar{\mathbf{e}}$. This program can be compiled and vectorized automatically. Alternatively we can use special FORTRAN 200 instructions such as Q8SSUM, which computes sums like Σe_{ir}. The computation of $\hat{\text{cov}}(\mathbf{y})$ in (3.4) can be programmed analogous to Figure 5. Alternatively we can program innerproducts ($\mathbf{e}'\mathbf{e}$ and $\bar{\mathbf{e}}\,\bar{\mathbf{e}}'$) through the special function Q8SDOT; see SARA (1984, pp. 22,30).

FIGURE 5
Vectorizable FORTRAN program for $\bar{e}$.

```
      DENOM = 1.0/m
      DO    10    I = 1,N
         DO    10    R = 1,M
            DO    10    LL = 1,L
10             EBAR(LL,I) = EBAR(LL,I) + E(LL,R,I)
      DO    20    I = 1,N
         DO    20    LL = 1,L
20             EBAR(LL,I) = EBAR(LL,I) * DNOM
```

3.2 A roadblock to vectorization

A problem arises when computing the *inverse* $[\hat{\mathbf{cov}}(\mathbf{y})]^{-1}$, which is needed to compute the EGLS estimator $\tilde{\underline{\beta}}$ in (2.7). The trick in the preceding steps was to make the inner loop long; that is, the LL loop became the inner loop. The instruction within that loop can be executed in parallel, provided that instruction contains *no function or subroutine references* except for basic functions such as sine: the vector computer can execute in parallel basic operations only; see SARA (1984, p. 23). So the computer cannot calculate L inverses in parallel, since calculating an inverse requires a subroutine call.

To invert a matrix we call a NAG routine (namely F01AAF, which solves linear equations using Crout's method). Obviously a subroutine call can always be avoided: replace the subroutine by the appropriate lines of code. Moreover, there is often more than one computational technique: matrices can be inverted in several ways; covariances can be computed in different ways [see (2.4) and (3.5)], and so on. However, subroutines are there to help the user; so most times the user will call upon a subroutine. This problem illustrates a more general problem: how much effort does the user want to spend on programming in order to fit the problem to a specific computer so that this computer runs faster?

So $[\hat{cov}(y)]^{-1}$ must be computed in scalar mode. Once this inverse is available, some matrix multiplications follow such as $[\hat{cov}(y)]^{-1}$ X. The share of the matrix inversion in the total computation time determines the gain to be obtained through vectorization. To quantify these ideas, we compute the OLS and the EGLS estimates for a number of cases, comparing a CYBER 205 and a VAX 8700.

4. Computational Tests

Into the OLS estimator of (2.6) we substitute

$$W = (X' X)^{-1} X' \tag{3.5}$$

and into the EGLS of (2.7) we substitute

$$V = (X' [\hat{cov}(y)]^{-1} X)^{-1} X' [\hat{cov}(y)]^{-1}. \tag{3.6}$$

W needs to be computed only once, but V is calculated L = 100 times since $\hat{cov}(y)$ changes every time. For these computations we select three cases, as follows. We use a regression metamodel for k factors accounting for all k(k-1)/2 two-factor interactions besides the overall mean and the k main effects; so Q = 1 + k + k(k-1)/2. The experimental design is a 2^{k-p} design with $n = 2^{k-p} \geq Q$. If k = 2 then Q = 4 and $n = 2^2 = 4$. If k = 4 then Q = 11 and $n = 2^4 = 16$. If k = 6 then Q = 22 and $n = 2^{6-1} = 32$ (with the generator $x_{i6} = x_{i1} x_{i2} x_{i3} x_{i4} x_{i5}$). We keep the number of simulation replicates at its minimum: m = n + 1 (if m ≤ n then $\hat{cov}(y)$ is singular). To improve the accuracy of our timing data we repeat the computation 100 times. Appendix 2 gives the main part of the computer program. This yields Table 3.

The CYBER 205 can run in vector mode and in scalar mode respectively. Table 3 shows that for OLS the scalar mode of this expensive computer runs only slightly faster than the VAX does. In vector mode, however, the CYBER takes less than 2% of the VAX time. In our EGLS code, matrix inversion cannot be vectorized. Therefore we measure how much time inversion takes. Obviously scalar mode and vector mode of the CYBER yield the same CPU times for inversion, apart from measurement errors. The "rest" in

Table 3 refers to the whole computer code excluding matrix inversion. In "vector" mode we vectorized all instructions that can be vectorized over the L dimension; see Appendix 2. In the small problem (n = 4, Q = 4) non-vectorizable inversion takes 85% of total time; consequently, vectorizing the rest can never save more than 15%; it does save 14%. In the large problem (n = 32, Q = 22) inversion takes only 28% of total time; vectorizing the rest saves 70%. Note that for the small problem, EGLS runs faster on the VAX than on the CYBER, even in vector mode. Appendix 3 gives some more programming tricks for improving the efficiency of supercomputers.

TABLE 3

Total CPU times (in microseconds)

(m=n+1, L=100)

OLS		n = 4 Q = 4	n = 16 Q = 11	n = 32 Q = 22
VAX 8700		710	7,870	29,060
CYBER: scalar mode		544	6,188	24,035
vector mode		11	123	486
EGLS				
VAX 8700	total	29,722	495,450	3,261,370
	inversion	22,550	243,150	1,230,120
	rest	7,172	252,300	2,031,250
CYBER: scalar	total	37,797	361,322	2,058,737
	inversion	32,267	168,244	584,393
	rest	5,530	193,078	1,474,344
CYBER: vector	total	32,437	172,854	625,000
	inversion	32,297	168,084	583,639
	rest	140	4,770	41,361

5. Conclusions

Vector computers provide a new challenge for management scientists, since their application requires a new way of thinking, namely "thinking in vector mode". This paper examined vector computing in Monte Carlo experiments with regression models used as metamodels in simulation. Then the matrix of independent variables **X** is relatively small, so vector computers are inefficient if applied straightforwardly. Monte Carlo experiments, however, are replicated many times, say 100 times; exploiting this dimension of the problem makes vector computers efficient in applications such as Ordinary Least Squares. Other applications such as Estimated Generalized Least Squares require subroutine calls; for example, matrix inversion. In small problems, vector computers such as the CYBER 205 are then slower than scalar computers such as the VAX 8700 are. So the researcher must estimate which fraction of the total computer time can be saved by vectorization. Moreover, exploiting vector computers requires researcher's time to figure out efficient implementations.

Appendix 1: FORTRAN 200 program for the pseudorandom number generator

```
      PROGRAM VARIANT4
      IMPLICIT REAL (U-Z), INTEGER (A-T)
      PARAMETER (N1=5,N4=65535,K=1)
      PARAMETER (A1=37772072706109)
      INTEGER MVAST
      BIT BVAST
      DESCRIPTOR MVAST, BVAST
      DIMENSION T(N4), S1(N1)
      DIMENSION X1(N1)
      DATA MINT / X'0000800000000000' /
      CALL RANSET (K)
      DO 5 I=1,N4
       U=RANF()
       CALL RANGET(T(I))
    5 CONTINUE  C ! N=5
C     ! SCALAR
      S1(1;N1)=T(1;N1)
      ZPU1=SECOND()
      DO 10 I=1,N1
       S1(I)=A1*S1(I)
       IF (S1(I).LT.0) S1(I)=S1(I)-MINT
       X1(I)=S1(I)/MINT
   10 CONTINUE
      ZPU2=SECOND()
      U1=ZPU2-ZPU1
C     ! VECTOR
      ASSIGN MVAST,.DYN.N1
      ASSIGN BVAST,.DYN.N1
      S1(1;N1)=T(1;N1)
      ZPU1=SECOND()
      S1(1;N1)=A1*S1(1;N1)
C     ! CONTROLLED INTEGER OVERFLOW
      BVAST=S1(1;N1).LT.0
      MVAST=S1(1;N1)-MINT
```

```
      S1(1;N1)=Q8VCTRL(MVAST,BVAST;S1(1;N1))
      X1(1;N1)=S1(1;N1)/MINT
      ZPU2=SECOND()
      Z1=ZPU2-ZPU1
      FREE
      PRINT *, 'BEGIN:  VECTORIZED SCALAR
      PRINT *, 'N=    5 '.Z1,'  ',U1
      END
```

Appendix 2: FORTRAN 200 program for the OLS and EGLS estimators

OLS ESTIMATOR FOR BETA

```
      CALL MXM(XT,X,XTX)
      CALL INVERSE(XTX,XTXI)
      CALL MXM(XTXI,XT,W)
       DO 5 I=1,N
        DO 5 J=1,M
         YGEM(1,I;LL)=YGEM(1,I;LL)+Y(1,J,I;LL)
 5    CONTINUE
      DO 10 I=1,R
       DO 10 J=1,N
        BETA(1,I;LL)=BETA(1,I;LL)+W(I,J)*YGEM(1,J;LL)
10    CONTINUE
```

EGLS ESTIMATOR FOR BETA

```
      DO 5 I=1,N
       DO 5 J=1,M
        YGEM(1,I;LL)=YGEM(1,I;LL)+Y(1,J,I;LL)
 5    CONTINUE
      DO 10 I=1,N
       DO 10 J=1,N
        DO 10 K=1,M
         S(1,I,J;LL)=S(1,I,J;LL)+((Y(1,K,I,;LL)-
      YGEM(1,I;LL))*(Y(1,K,J;LL)-YGEM(1,J;LL)))
```

```
10    CONTINUE
      DO 15 I=1,
       DO 15 J=1,N
        S(1,J,I;LL)=S(1,I,J;LL)
15    CONTINUE
      DO 18 K=1,LL
       DO 20 I=1,N
        DO 20 J=1,N
         DU4(J,1)=S(K,J,I)
20     CONTINUE
       CALL INVERSE(DU4,MY4,N)
       DO 25 I=1,N
        DO 25 J=1,N
         SI(K,J,I)=MY4(J,I)
25     CONTINUE
18    CONTINUE
      DO 30 I=1,R
       DO 30 J=1,N
        DO 30 K=1,N
         XTSI(1,I,J;LL)=XTSI(1,I,J;LL)+XT(I,K*SI(1,K,J;LL)
30    CONTINUE
      DO 35 I=1,R
       DO 35 J=1,R
        DO 35 K=1,N
       XTSIX(1,I,J;LL)=XTSIX(1,I,J;LL)+XTSI(1,I,K;LL)*X(K,J)
35    CONTINUE
       DO 40 K=1,LL
        DO 45 I=1,R
         DO 45 J=1,R
          DU3(J,I)=XTSIX(K,J,I)
45       CONTINUE
        CALL INVERSE(DU3,MY3,R)
        DO 50 I=1,R
        DO 50 J=1,R
         XTSIXI(K,J,I)=MY3(J,I)
50      CONTINUE
```

```
40    CONTINUE
      DO 55 I=1,R
       DO 55 J=1,N
        DO 55 K=1,R
       V(1,I,J;LL)=V(1,I,J;LL)+XTSIXI(1,I,K;LL)*XTSI(1,K,J;LL)
55    CONTINUE
      DO 60 I=1,R
       DO 60 K=1,N
        BETA(1,I;LL)=BETA(1,I;LL)+V(1,I,K;LL)*YGEM(1,K;LL)
60    CONTINUE
```

Appendix 3: Programming tricks

There are several "tricks" for improving the efficiency of vector computers. These tricks should be applied in any computer program, not only Monte Carlo experiments:

1. Scalar divides take relatively much time (54 cycles versus 5 cycles for multiplication; 1 cycle takes 20 nanoseconds); the computation of denominators like 1/m (see Figure 5) and 1/(m-1) (see eq. 3.4) should therefore be separated by several lines of code; SARA (1984, pp. 5,7).
2. Double precision is slow and excludes vector mode; SARA (1984, p. 6).
3. There are special vectorized instructions so-called V-functions and Q8-functions. We presented some examples; also see SARA (1984, pp. 27,30).
4. The compiler can optimize the standard FORTRAN program; next special programs (such as SPY and CIA) can measure which parts of the program take most time during execution and are candidates for customized optimization.

Acknowledgement

The first author was sponsored by the Supercomputer Visiting Scientist Program at Rutgers University, The State University of New Jersey, during July 1988. In 1989/1990 computer time on the CYBER 205 in Amsterdam was made available by SURF/NFS. The editor (Jim Wilson) and three anonymous referees gave many comments that lead to an expanded and better organized paper, and to the elimination of a few errors; any remaining errors are the authors' responsibility.

References

CDC, *FORTRAN 200 Version 1 Reference Manual*, Publicatio no. 60480200, Control Data Corporation, Sunyvale, California 94088-3492, December 1986.

Devroye, L., *Non-Uniform Random Variate Generation*, Springer-Verlag, New York, 1986.

Dijkstra, R.L., "Establishing the Positive Definiteness of the Sample Covariance Matrix," *The Annals of Mathematical Statistics*, 41 (1970), 2153-2154.

Heidelberger, P., "Discrete Event Simulations and Parallel Processing: Statistical Properties," *SIAM J. Stat. Comput.*, 39 (1988), 1114-1132.

Kleijnen, J.P.C., *Statistical Tools for Simulation Practitioners*, Marcel Dekker, Inc., New York, 1987.

Kleijnen, J.P.C., "Analyzing Simulation Experiments with Commmon Random Numbers," *Management Sci.*, 34 (1988), 65-74.

Kleijnen, J.P.C., "Pseudorandom Number Generation on Supercomputers," *Supercomputer*, 6 (1989), 34-40.

Kleijnen, J.P.C., *Regression Metamodels for Simulation with Common Random Numbers: Comparison of Techniques*, Katholieke Universiteit Brabant (Tilburg University), January 1990. (Submitted for publication.)

Kleijnen, J.P.C. and B. Annink, *Multiplicative Congruential Generators for Supercomputers and Other Computers*, Katholieke Universiteit Brabant (Tilburg University), Oct. 1989. (Submitted for publication.)

Levine, R.D., "Supercomputers," *Scientific American*, (1982), 112-125.

Naylor, T.H., J.L. Balintfy, D.S. Burdick and K. Chu, *Computer Simulation Techniques*. Wiley, New York, 1966.

Neely, P.M.,"Comparison of Several Algorithms for Computation of Means, Standard Deviations, and Correlation Coefficients", *Communications of the ACM*, 9 (1966), 496-499.

Park, S.K. and Miller, K.W., "Random Number Generators: Good Ones are Hard to Find," *Communications of the ACM*, 31 (1988), 1192-1201.

Petersen, W.P., "Some Vectorized Random Number Generators for Uniform, Normal, and Poisson Distributions for CRAY X-MP", *The Journal of Supercomputing*, 1 (1988), 327-335.

SARA, *Cyber 205 User's guide; part 3, Optimization of FORTRAN programs*. SARA (Stichting Academisch Rekencentrum Amsterdam/ Foundation Academic Computer Centre Amsterdam), Amsterdam, 1984.

Teichroew, D., "A History of Distribution Sampling Prior to the Era of the Computer and its Relevance to Simulation," *J. Am. Stat. Ass.*, 1965, 27-49.

IN 1989 REEDS VERSCHENEN

368 Ed Nijssen, Will Reijnders
"Macht als strategisch en tactisch marketinginstrument binnen de distributieketen"

369 Raymond Gradus
Optimal dynamic taxation with respect to firms

370 Theo Nijman
The optimal choice of controls and pre-experimental observations

371 Robert P. Gilles, Pieter H.M. Ruys
Relational constraints in coalition formation

372 F.A. van der Duyn Schouten, S.G. Vanneste
Analysis and computation of (n,N)-strategies for maintenance of a two-component system

373 Drs. R. Hamers, Drs. P. Verstappen
Het company ranking model: a means for evaluating the competition

374 Rommert J. Casimir
Infogame Final Report

375 Christian B. Mulder
Efficient and inefficient institutional arrangements between governments and trade unions; an explanation of high unemployment, corporatism and union bashing

376 Marno Verbeek
On the estimation of a fixed effects model with selective non-response

377 J. Engwerda
Admissible target paths in economic models

378 Jack P.C. Kleijnen and Nabil Adams
Pseudorandom number generation on supercomputers

379 J.P.C. Blanc
The power-series algorithm applied to the shortest-queue model

380 Prof. Dr. Robert Bannink
Management's information needs and the definition of costs, with special regard to the cost of interest

381 Bert Bettonvil
Sequential bifurcation: the design of a factor screening method

382 Bert Bettonvil
Sequential bifurcation for observations with random errors

383 Harold Houba and Hans Kremers
Correction of the material balance equation in dynamic input-output models

384 T.M. Doup, A.H. van den Elzen, A.J.J. Talman
Homotopy interpretation of price adjustment processes

385 Drs. R.T. Frambach, Prof. Dr. W.H.J. de Freytas
Technologische ontwikkeling en marketing. Een oriënterende beschouwing

386 A.L.P.M. Hendrikx, R.M.J. Heuts, L.G. Hoving
Comparison of automatic monitoring systems in automatic forecasting

387 Drs. J.G.L.M. Willems
Enkele opmerkingen over het inversificerend gedrag van multinationale ondernemingen

388 Jack P.C. Kleijnen and Ben Annink
Pseudorandom number generators revisited

389 Dr. G.W.J. Hendrikse
Speltheorie en strategisch management

390 Dr. A.W.A. Boot en Dr. M.F.C.M. Wijn
Liquiditeit, insolventie en vermogensstructuur

391 Antoon van den Elzen, Gerard van der Laan
Price adjustment in a two-country model

392 Martin F.C.M. Wijn, Emanuel J. Bijnen
Prediction of failure in industry
An analysis of income statements

393 Dr. S.C.W. Eijffinger and Drs. A.P.D. Gruijters
On the short term objectives of daily intervention by the Deutsche Bundesbank and the Federal Reserve System in the U.S. Dollar - Deutsche Mark exchange market

394 Dr. S.C.W. Eijffinger and Drs. A.P.D. Gruijters
On the effectiveness of daily interventions by the Deutsche Bundesbank and the Federal Reserve System in the U.S. Dollar - Deutsche Mark exchange market

395 A.E.M. Meijer and J.W.A. Vingerhoets
Structural adjustment and diversification in mineral exporting developing countries

396 R. Gradus
About Tobin's marginal and average q
A Note

397 Jacob C. Engwerda
On the existence of a positive definite solution of the matrix equation $X + A^T X^{-1} A = I$

398 Paul C. van Batenburg and J. Kriens
Bayesian discovery sampling: a simple model of Bayesian inference in auditing

399 Hans Kremers and Dolf Talman
Solving the nonlinear complementarity problem

400 Raymond Gradus
Optimal dynamic taxation, savings and investment

401 W.H. Haemers
Regular two-graphs and extensions of partial geometries

402 Jack P.C. Kleijnen, Ben Annink
Supercomputers, Monte Carlo simulation and regression analysis

403 Ruud T. Frambach, Ed J. Nijssen, William H.J. Freytas
Technologie, Strategisch management en marketing

404 Theo Nijman
A natural approach to optimal forecasting in case of preliminary observations

405 Harry Barkema
An empirical test of Holmström's principal-agent model that tax and signally hypotheses explicitly into account

406 Drs. W.J. van Braband
De begrotingsvoorbereiding bij het Rijk

407 Marco Wilke
Societal bargaining and stability

408 Willem van Groenendaal and Aart de Zeeuw
Control, coordination and conflict on international commodity markets

409 Prof. Dr. W. de Freytas, Drs. L. Arts
Tourism to Curacao: a new deal based on visitors' experiences

410 Drs. C.H. Veld
The use of the implied standard deviation as a predictor of future stock price variability: a review of empirical tests

411 Drs. J.C. Caanen en Dr. E.N. Kertzman
Inflatieneutrale belastingheffing van ondernemingen

412 Prof. Dr. B.B. van der Genugten
A weak law of large numbers for m-dependent random variables with unbounded m

413 R.M.J. Heuts, H.P. Seidel, W.J. Selen
A comparison of two lot sizing-sequencing heuristics for the process industry

414 C.B. Mulder en A.B.T.M. van Schaik
Een nieuwe kijk op structuurwerkloosheid

415 Drs. Ch. Caanen
De hefboomwerking en de vermogens- en voorraadaftrek

416 Guido W. Imbens
Duration models with time-varying coefficients

417 Guido W. Imbens
Efficient estimation of choice-based sample models with the method of moments

418 Harry H. Tigelaar
On monotone linear operators on linear spaces of square matrices

IN 1990 REEDS VERSCHENEN

419 Bertrand Melenberg, Rob Alessie
A method to construct moments in the multi-good life cycle consumption model

420 J. Kriens
On the differentiability of the set of efficient (μ,σ^2) combinations in the Markowitz portfolio selection method

421 Steffen Jørgensen, Peter M. Kort
Optimal dynamic investment policies under concave-convex adjustment costs

422 J.P.C. Blanc
Cyclic polling systems: limited service versus Bernoulli schedules

423 M.H.C. Paardekooper
Parallel normreducing transformations for the algebraic eigenvalue problem

424 Hans Gremmen
On the political (ir)relevance of classical customs union theory

425 Ed Nijssen
Marketingstrategie in Machtsperspectief

426 Jack P.C. Kleijnen
Regression Metamodels for Simulation with Common Random Numbers: Comparison of Techniques

427 Harry H. Tigelaar
The correlation structure of stationary bilinear processes

428 Drs. C.H. Veld en Drs. A.H.F. Verboven
De waardering van aandelenwarrants en langlopende call-opties

429 Theo van de Klundert en Anton B. van Schaik
Liquidity Constraints and the Keynesian Corridor

430 Gert Nieuwenhuis
Central limit theorems for sequences with m(n)-dependent main part

431 Hans J. Gremmen
Macro-Economic Implications of Profit Optimizing Investment Behaviour

432 J.M. Schumacher
System-Theoretic Trends in Econometrics

433 Peter M. Kort, Paul M.J.J. van Loon, Mikulás Luptacik
Optimal Dynamic Environmental Policies of a Profit Maximizing Firm

434 Raymond Gradus
Optimal Dynamic Profit Taxation: The Derivation of Feedback Stackelberg Equilibria

435 Jack P.C. Kleijnen
Statistics and Deterministic Simulation Models: Why Not?

436 M.J.G. van Eijs, R.J.M. Heuts, J.P.C. Kleijnen
Analysis and comparison of two strategies for multi-item inventory systems with joint replenishment costs

437 Jan A. Weststrate
Waiting times in a two-queue model with exhaustive and Bernoulli service

438 Alfons Daems
Typologie van non-profit organisaties

439 Drs. C.H. Veld en Drs. J. Grazell
Motieven voor de uitgifte van converteerbare obligatieleningen en warrantobligatieleningen

440 Jack P.C. Kleijnen
Sensitivity analysis of simulation experiments: regression analysis and statistical design

441 C.H. Veld en A.H.F. Verboven
De waardering van conversierechten van Nederlandse converteerbare obligaties

442 Drs. C.H. Veld en Drs. P.J.W. Duffhues
Verslaggevingsaspecten van aandelenwarrants